INSTRUCTION MEMORY HIERARCHY FOR AN EMBEDDED PROCESSOR

Inventors: Donald E. Steiss

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

[0001] This invention relates generally to a computer processor, and more specifically, to a network processor having an instruction memory hierarchy that distributes instructions to a plurality of processing units organized in clusters within the network processor.

BACKGROUND ART

[0002] Until recently, a lack of network bandwidth posed restraints on network performance. But emerging high bandwidth network technologies now operate at rates that expose limitations within conventional computer processors. Even high-end network devices using state of the art general purpose processors are unable to meet the demands of networks with data rates of 2.4-Gbs, 10-Gbs, 40-Gbs and higher.

[0003] Network processors are a recent attempt to address the computational needs of network processing which, although limited to specialized functionalities, are also flexible enough to keep up with often changing network protocols and architecture. Compared to general processors performing a variety of tasks, network processors primarily perform packet processing tasks using a relatively small amount of software code. Examples of specialized packet processing include packet routing, switching, forwarding, and bridging. Some network processors have arrays of processing units with

multithreading capability to process more packets at the same time. However, current network processors have failed to address certain characteristics of network processing by relying too much on general processing architectures and techniques.

[0004] Access to instructions is one problem associated with a processing array having a conventional memory architecture. During an instruction fetch stage in a processing unit pipeline, each processing unit retrieves software code for execution during an execution stage from a memory element. In some typical processing arrays, each processing unit has dedicated instruction memory. But dedicated memory consumes valuable die area on the processor and inefficiently replicates the same code. In other typical processing arrays, processing units share a common instruction memory. But competition for memory access between processing units increases latencies during an instruction fetch pipeline stage. Furthermore, memory has limited bandwidth that is not capable of delivering instructions at a rate required by a processing array performing packet processing.

[0005] Limited instruction memory bandwidth causes more severe problems in processor arrays with multithreaded processing units. In hardware-level multithreading, a logically-partitioned processor streams instructions through its pipeline for more than one hardware thread at the same time to improve effective CPIs (Cycles Per Instruction). Each hardware thread can be associated with a different application program or network packet. When one thread experiences a stall caused by, for example, a memory access latency during the execution stage, the processing unit switches execution to a different thread rather than wasting execution cycles.

[0006] By contrast, in software-level multithreading, a single application program streams instructions to the processor using several software threads or processes. "Multithreading" and "threads" as used herein, however, refer to hardware multithreading and hardware instruction threads, respectively. Because multithreaded processing units have higher CPIs (i.e., instructions processed each cycle), even more instruction memory bandwidth is needed for instruction fetches. Moreover, since each thread has an independent instruction stream, there is even more contention for available memory bandwidth.

[0007] Therefore, what is needed is a processor including an instruction memory hierarchy and method of distributing instructions to an array of multithreaded processing units. Furthermore, there is a need for an instruction request arbiter and method for controlling instruction requests from the array of multithreaded processing units to the instruction memory hierarchy.

SUMMARY OF THE INVENTION

[0008] The present invention meets these needs by providing a processor with an instruction memory hierarchy and method for distributing instructions to an array of multithreaded processing units. In one embodiment, the instruction memory hierarchy comprises a plurality of processing units, an instruction request arbiter, and an instruction memory. A signal line, such as an instruction request bus, coupled to outputs of the plurality of processing units and to an input of the instruction request arbiter, sends signals representing instruction requests. A signal line, coupled to a first output of the instruction memory arbiter and to an input of the instruction memory, forwards signals

representing non-conflicting instruction requests. A signal line, coupled to an output of the instruction memory and to inputs of the plurality of processing units, sends signals representing instruction data. The instruction request arbiter receives instruction requests comprising requests to the instruction memory to fill processor instruction cache sublines. A first embodiment of the instruction request arbiter controls instruction request submissions from each of the plurality of processing units, and also resolves conflicts between instruction requests. The instruction memory broadcasts instruction data responsive to non-conflicting instruction requests forwarded from the instruction request arbiter.

[0009] A second embodiment of the instruction request arbiter controls instruction request submissions from the plurality of processors with a traffic mode. The instruction request bus comprises a pipelined slotted ring that couples a second output of the instruction request arbiter to second inputs of the plurality of processors and the instruction arbiter input. The instruction request arbiter broadcasts a high traffic mode signal to the plurality of processors responsive to an average amount of instruction requests exceeding a limit. During high traffic mode, the plurality of processing units send instruction requests at scheduled times. The instruction request arbiter also broadcasts a low traffic mode signal to the plurality of processing units responsive to the average amount of instruction requests being within a limit. During low traffic mode, the plurality of processing units send instruction requests during empty slots.

[0010] A third embodiment of the instruction request arbiter resolves conflicts between instruction requests based on instruction priorities. Instruction requests comprise one or more priority bits whereby, for example, a high priority indication is

associated with a critical instruction request. The instruction request arbiter resolves conflicts between instruction requests having different priorities (e.g., in an embodiment with two-level priority having a high priority instruction request and a low priority instruction request) by selecting the high priority instruction request. The instruction request arbiter increments, and possibly saturates, the priority of the unselected instruction request to a higher priority instruction request before recirculating it around the slotted ring. The instruction request arbiter resolves conflicts between instructions having the same priority (e.g., a first high priority request and a second high priority request) by using round-robin arbitration. The unselected request is incremented if low priority, and recirculated.

BRIEF DESCRIPTION OF THE FIGURES

[0011] FIG. 1 is a block diagram illustrating a network processing system according to one embodiment of the present invention.

[0012] FIG. 2 is a block diagram illustrating the processor according to one embodiment of the present invention.

[0013] FIG. 3 is a block diagram illustrating an instruction memory hierarchy according to one embodiment of the present invention.

[0014] FIG. 4 is a block diagram illustrating the instruction request arbiter according to one embodiment of the present invention.

[0015] FIG. 5 is a flow chart illustrating a method for controlling instruction requests in the instruction request arbiter according to one embodiment of the present

invention.

[0016] FIG. 6 is a flow chart illustrating the method for determining the traffic mode according to one embodiment of the present invention.

[0017] FIG. 7 is a flow chart illustrating the method of resolving conflicts between instruction requests according to one embodiment of the present invention.

DETAILED DESCRIPTIONS OF THE PREFERRED EMBODIMENTS

I. Introduction

[0018] Many variations will be apparent to one of ordinary skill in the art that would yet be encompassed by the spirit and scope of the invention. For example, although the present invention provides optimum performance to a network processing system, it may be implemented in any many other environments such as in a supercomputer, a personal computer, a workstation, a PDA (Personal Digital Assistants), a digital signal processing system, or the like. Accordingly, the below description is included to illustrate the operation of the preferred embodiments and is not meant to limit the scope of the invention. Rather, the scope of the invention is to be limited only by the claims that follow.

II. Network Processing System

[0019] FIG. 1 is a block diagram illustrating a network processing system 100 according to one embodiment of the present invention. The system 100 comprises a processor 110, external memory 120, and a co-processor 130 each coupled by signal line (or bus) 102. The system 100 may be a specialized computing device such as a router, a

switch, a bridge, a gateway, or a combination of devices such as the 12000-series systems manufactured and sold by Cisco Systems, Inc. of Sunnyvale, CA. The system 100 processes incoming packets 140 received from a network (not shown) resulting in outgoing packets 150 sent through the network. More specifically, the processor 110 operates in conjunction with the co-processor 130 to perform various tasks such as routing, switching, bridging, and packet forwarding using various network protocols such as TCP/IP (Transmission Control Protocol / Internet Protocol), ATM (Asynchronous Transfer Mode), IEEE 802.3, IEEE 802.11, etc. The system 100 operates to service high-speed networks with bandwidths of 2.4-Gbs, 10-Gbs, 40-Gbs, and above. In one embodiment, an operating system such as VxWorks by WindRiver of Alameda, CA or other controlling software manages interactions between the hardware and software. One of ordinary skill in the art will recognize various configurations of the network device 110 within the scope of the present invention.

[0020] The processor 110 executes packet processing instructions to process the incoming packets 140. The processor 110 comprises, for example, an x86-type processor, a network processor, a multithreaded processor, a multiple instruction multiple data processor, a general processing unit, an application specific integrated circuit, or any processing device capable of processing instructions. The processor 110 is implemented as, for example, an integrated circuit on substrates such as silicon or geranium, a field programmable device, a nanotechnology-based chip, or any other type of chip for implementing logic functionality at high data rates. In one embodiment, the processor 110 is implemented as described with reference to FIGs. 2-4 performing methods described with reference to FIGs. 5-7.

[0021] Packet processing instructions comprise a set of general purpose instructions, packet processing extensions, or other software code that is executed by the processor 110. Examples of packet processing extensions include bit field manipulation instructions, checksum acceleration instructions, and encryption instructions. In one embodiment, at least some instructions include an external priority set by a programmer or other software such as during critical packet processing. The external priority influences arbitration between conflicting instruction requests as described below.

[0022] The external memory 120 provides large capacity, albeit long latency, storage for packet processing instructions 120. Preferably, the external memory 120 is limited to storing rarely used code. The external memory 120 comprises, for example, SRAM, EEPROM, or any other memory element capable of storing packet processing instructions. Note that instruction fetches to the external memory 120 by the processor 110 may cause latencies and thread stalls. In one implementation, the external memory 120 contains 128MB of addressable space. Operation of the external memory 120 within the memory hierarchy is described below with reference to FIG. 3.

[0023] The co-processor 130 performs processing tasks to support the processor 110. The co-processor 130 comprises, for example, a CAM (Content Addressable Memory) device to provide ACL (Access Control List) look-ups, a search engine to provide packet forwarding tasks, a buffer to store incoming and outgoing packets 130, 140, and the like.

[0024] FIG. 2 is a block diagram illustrating the processor 110 according to one embodiment of the present invention. The processor 110 comprises an instruction

memory 230, an instruction request arbiter 220, and processor clusters 210a-d, and an instruction request bus 225 (collectively referring to signal lines 202a-f and flip flops 215a-e).

[0025] The instruction request bus 225 is coupled to first inputs and outputs of the processor clusters 210a-d and to an input of the instruction request arbiter 220, and signal lines 222 are coupled to a first output of the instruction request arbiter 220 and to an input of the instruction memory 230. Signal line 212 is coupled to a first output of the instruction memory 230 and to second inputs of the processor clusters 210a-d. Signal line 102 is coupled to a second output of the instruction memory 230 to send signals to the external memory 120. Additionally, signal line 203 is coupled to an input of the processor 110 and the input of the instruction request arbiter 220, and signal line 223 is coupled to a third output of the instruction request arbiter 220 and a first output of the processor. Signal line 213 is also coupled to the first output of the instruction memory 230 and a second output of the processor 110. Signal line 102 is coupled to an input/output of the instruction memory 230 and the input/output of the processor 110.

[0026] In general, the instruction request arbiter 220 receives and forwards instruction requests from the processor clusters 210a-d to the instruction memory 230 through the instruction request bus 225. In response to a forwarded instruction request, the instruction memory 230 broadcasts a corresponding instruction to the processor clusters 210a-d. In one embodiment, an off-chip instruction request bus coupled to off-chip processor clusters (not shown) is coupled to signal lines 203, 223, 213 to access instructions from the multi-memory bank 230. One or ordinary skill in the art will recognize additional configurations of processor clusters 210a-d and off-chip processor

clusters within the scope of the present invention.

[0027] The processor clusters 210a-d comprise one or more processing units that share access to the instruction request bus 225. The processor clusters 210a-d access the instruction request bus 225 through signal lines 202b-e, respectively, to send instruction requests according to either the high or low traffic mode policy. In high traffic mode, the processor clusters 210a-d submit instruction requests according to a globally synchronized counter (not shown). The processor clusters 210a-d receive corresponding instruction data by snooping, i.e., testing all instruction broadcasts from the instruction memory 230 on signal line 212 against currently outstanding instruction requests in the processor cluster 210. The processor clusters 210a-d are described further below with reference to FIG. 3.

[0028] The instruction request bus 225 transports instruction requests from the processor clusters 210a-d to the instruction request arbiter 220. The instruction request bus 225 comprises: signal line 202f coupled to the second output of the instruction request arbiter 220 and an input of flip flop 215e; signal line 202e coupled to an output of flip flop 215e and an input of flip flop 215d; signal line 202d coupled to an output of flip flop 215d and an input of flip flop 215c; signal line 202c coupled to an output of flip flop 215c and an input of flip flop 215b; signal line 202b, coupled to an output of flip flop 215b and an input of flip flop 215a; and signal line 202a coupled to an output of flip flop 215a and the input of the instruction request arbiter 220. Flip flops 215a-e store bits representing instruction requests (or lack of) and traffic modes. In another embodiment, the instruction request bus 225 comprises buffers, registers, or other memory elements for pipelining. The instruction request bus 225 comprises, for example, a pipelined slotted

ring bus. An instruction request format comprises, for example, bit $_0$ indicating validity; bits $_{1\text{-}2}$ indicating priority; bit $_3$ containing a critical subline flag; bits $_{4\text{-}30}$ containing a cache subline address; and bit $_{31}$ indicating a traffic mode.

[0029] The instruction request arbiter 220 controls access to the instruction request bus 225 with traffic modes. The instruction request arbiter 220 sets a traffic mode on the instruction request bus 225 as either high or low depending on an average number of instruction requests. Flip-flops 215a-e send the traffic mode, or control bit, to the processor clusters 210a-d. In low traffic mode, empty instruction request bits, an invalid instruction request flag, a low traffic mode bit, or other indicator from the flip-flops 215a-e signify that an instruction request can be submitted by a processor cluster 210. In high traffic mode, the flip-flops 215a-e provide synchronizing information, such as a count or counter reset, to the processor clusters 210a-d. Each processor cluster 210, in the count example, submits instructions requests at an assigned offset from the count.

[0030] The instruction request arbiter 220 also controls access to the instruction memory by selecting non-conflicting instruction requests to forward. Conflicting instruction requests require access to a common group of cache sets of the instruction memory 230. The instruction request arbiter 220 resolves conflicting instruction requests using either priority levels associated with the instruction requests or round-robin arbitration. In one embodiment, the instruction request arbiter 220 accepts instruction requests from more than one instruction request bus 225, and forwards more than one instruction request to the instruction memory 230. The instruction request arbiter 220 and related methods are described below in more detail.

[0031] The instruction memory 230 stores packet processing instructions executed by the processor clusters 210a-d. The instruction memory 230 responds to non-conflicting instruction requests forwarded by the instruction request arbiter 220 by broadcasting corresponding instruction data on one or both of signal lines 212, 214. Instruction requests are read requests of an address potentially stored in the instruction memory 230. Instruction requests containing addresses not found in the instruction memory 230 are sent to the external memory 120 through signal line 102. Preferably, the instruction memory 230 is clocked at a core rate. The instruction memory 230 comprises, for example, a set associative instruction cache, a multiple bank memory, an SRAM, a DRAM, or any other memory element capable or storing instructions. In one implementation, the instruction memory 230 is 128kB and 8 way banked with 16 cache sets per bank referred to as a group of cache sets. The instruction memory 230 may operate at the core clock rate. A read width is a subline that may be 32 bytes.

[0032] In one embodiment, signal lines 212, 213 are, for example, result buses, fill buses, or any signal lines capable of broadcasting instructions. Preferably, signal lines 212, 213 are clocked at the core rate. As described, processor clusters 210a-d retrieve instruction data from signal line 212 matching posted instruction requests. If instruction data does not match any posts, as in the case of a branch mispredictions or exceptions, the instruction data is ignored. In one example, the signal lines 212, 213 have a width of 285 bits with bit $_0$ indicating instruction validity, bits $_{1\text{-}27}$ containing a cache subline address, bits $_{28\text{-}283}$ containing subline data, and bit $_{284}$ containing a parity bit.

[0033] In another embodiment the instruction memory 230 comprises a cache control mechanism (not shown) to manage which instructions are stored. Accordingly,

an instruction address can be: non-cached in the processor cluster 210, but cached in the instruction memory 230; non-cached in the processor cluster 210 and non-cached in the instruction memory 230; cached in the processor cluster 210 and cached in the instruction memory 230; or cached in the processor cluster 210, but non-cached in the instruction memory 230.

III. Instruction Memory Hierarchy

[0034] FIG. 3 is a block diagram illustrating an instruction memory hierarchy 300 of according to one embodiment of the present invention. The memory hierarchy 300 comprises a processor cluster 210 including a plurality of processing units 310a-d with caches 312a-d, the instruction request arbiter 220, the instruction memory 230, and the external memory 120. Signal line 302 couples outputs of the processing units 310a-d to an input/output of the processor cluster 210. The instruction request bus 225 couples the input/output of the processor cluster 210, to the input of the instruction request arbiter 220. As described above, the instruction request arbiter 220 is coupled to the instruction memory 230, which in turn, is coupled to the external memory 120. Also, the instruction memory 230 is coupled to an input of the processor cluster 210.

[0035] An instruction request from one of the processing units moves up the memory hierarchy 300 until there is a matching instruction address. More specifically, if there is a cache subline miss in a processor's cache 312, the instruction request moves to the instruction memory 230, and if the instruction memory 230 experiences a miss, the instruction request moves to the external memory 230. One of ordinary skill in the art will recognize that variations of the memory hierarchy 300, such as additional levels of memory contained within each processing unit 310a-d, are within the scope of the present

invention.

[0036] The processor cluster 210 comprises four processing units 310a-d. However, one of ordinary skill will recognize that the processor cluster 210 can comprise any number of processing units 310. The processing units 310a-d processes packet processing instructions related to packet routing, switching, bridging, and forwarding. The processing units 310a-d comprise, for example, a processing core, a network processing core, a multiple instruction multiple data core, a parallel processing element, a controller, or any other pipelined device for processing instructions. In an instruction fetch stage, the processing units 310a-d retrieve packet processing or other instructions from the memory hierarchy 300. The processing units 310a-d decodes the packet processing instructions and executes decoded instructions in an execution stage. In one embodiment, the processing units 310a-d comprise one or more multithreaded processing units capable of processing several threads through its pipeline at the same time. In this embodiment, the processing units 310a-d process multiple instruction threads, T_m^k, where, for example, $m = 2, 4, 128$ or more multithreaded instructions and k = the k^{th} instruction from thread m. Multithreaded processing units are described in greater detail in U.S. Patent Application No. <##/###,###> which is incorporated herein by reference in its entirety. In the multithreaded embodiment, each thread within a processing unit 310 submits instruction requests responsive to a cache miss in the processor's cache 312.

[0037] In one embodiment, the multithreaded processing units 310a-d comprise a bus interface (not shown) to manage accesses to the instruction request bus 225. The bus interface includes a counter to keep track of slots dedicated to the multithreaded processing unit 310a-d during high traffic mode. In high traffic mode, the bus interface

selects an instruction request, for example, every 16 cycles. In low traffic mode, the bus interface can select an instruction request whenever a neighboring processor cluster 210 had an unused time slot. The bus interface prioritizes instruction requests by priority and arrival time. The bus interface selects the lowest numbered request with the smallest timestamp for the next subline request.

[0038] In one embodiment, a cluster instruction cache (not shown) is shared by the processing units 310a-d, to provide low latency storage of recently accessed code. The cluster instruction cache receives instruction requests during the fetch stage of a processing unit 310a-d responsive to cache misses in the processor's cache 312. If the requested address is found, the cluster instruction cache returns associated instruction data. However, if there is a cache miss, the cluster instruction cache sends a request to the instruction memory 230. The cluster instruction cache listens to or snoops on signal line for corresponding instruction data broadcast by the instruction memory 230. In one implementation, the cluster instruction cache capacity is 16kB.

[0039] The instruction memory 230, shared by the processor clusters 210a-d, provides medium latency storage for preferably most of the code. The external memory 120, shared by all processing units 310 on the processor 110, provides high latency storage for preferably rarely used code.

[0040] FIG. 4 is a block diagram illustrating the instruction request arbiter 220 according to one embodiment of the present invention. The instruction request arbiter 220 comprises a bus access module 410 and a memory access module 420. The instruction request buses are coupled to an input of the bus access module 410 and an

input of the memory access module 420. Signal lines 202a, 203 are coupled to the input of the instruction request arbiter 220 and inputs of the bus access module 410 and the memory access module 420. Signal lines 222 are coupled to a first output of the memory access module 420 and the first output of the instruction request arbiter. Signal lines 202f, 223 are coupled to a second output of the bus access module 410, an output of the memory access module 420 and to the first output of the instruction request arbiter.

[0041] The bus access module 410 controls access of the processor clusters 210a-d to the instruction request bus 225 by determining a traffic mode. The bus access module 410 computes an average number of instruction requests during a period of time. The bus access module 410 broadcasts a high traffic mode responsive to the instruction request average exceeding a limit and a low traffic mode responsive to the instruction request average within the limit. In one example, if there are 15 or more instruction requests over a 64-clock cycle interval, the instruction request arbiter 220 sets the traffic mode to high. One of ordinary skill in the art will recognize various other traffic modes and mode detection methods within the scope of the present invention. The bus access module 410 and related methods are described in more detail below with reference to FIG. 6.

[0042] The memory access module 420 controls access to the instruction memory 230 by detecting and resolving conflicting instruction requests. The memory access module 420 detects conflicting instruction requests that attempt a read and/or write operation to the same memory bank within the instruction memory 230. To resolve conflicts between instruction requests having the same priority level (e.g., a first high priority instruction request and a second high priority instruction request), the memory

access module 420 uses round-robin arbitration. For example, the memory access module 420 can use a counter with representations for each of the processing units 310. The instruction request comprises an identification associated with a requesting processing unit 310. Whichever requesting processing unit 310 is next in line from the current count, is selected. One of ordinary skill in the art will recognize that, although the instruction request arbiter 220 is shown in FIG. 4 as receiving instruction requests from two instruction request busses 225, variations are within the scope of the present invention. Similarly, the instruction request arbiter 220 is shown in FIG. 4 as outputting two non-conflicting instruction requests to the instruction memory 230 may vary.

[0043] If the instruction requests have different priority levels (e.g., in a two-level priority embodiment, a high priority instruction request and a low priority instruction request), the memory access module 420 selects the instruction request with higher priority. A critical instruction request is one example of a high priority request. More specifically, the processing unit 310 can designate the first access of a multiple-access transaction, which contains an address with dependencies blocking program execution as a critical instruction request. Additionally, the memory access module 420 increments, and possibly saturates, the priority level of the losing instruction request from low to high prior to recirculating the instruction request around the instruction request buses. One of ordinary skill in the art will recognize varying priority schemes with more than two priority levels within the scope of the present invention. The memory access module 420 and related methods are described in more detail below with reference to FIG. 7.

[0044] FIG. 5 is a flow chart illustrating a method 500 for controlling instruction requests in the instruction request arbiter 220 according to one embodiment of the present

invention. The bus access module 410 controls 510 access to the instruction request bus 225 by determining a traffic mode as described below with reference to FIG. 6. The memory access module 420 controls 520 access to the instruction memory 230 by resolving conflicts between instruction requests as described below with reference to FIG. 7.

[0045] FIG. 6 is a flow chart illustrating the method 510 for determining the traffic mode of the instruction request bus 225 according to one embodiment of the present invention. One of ordinary skill in the art will recognize variations for determining the traffic mode within the scope of the present invention. In FIG. 6, the bus access module 410 computes 610 an average amount of instruction requests over a period of time. In one embodiment, the bus access module 410 counts the number of instruction requests during *n* cycles for *n* processing units.

[0046] If the average exceeds 620 a limit, the bus access module 410 broadcasts a high traffic mode to the processor clusters 210a-d. In one embodiment, the limit is exceeded when there are *n* instruction requests during *m* cycles, and *n* is, for example, within 10% of *m*. In another embodiment, high traffic mode is detected when the number of cycles between an oldest and newest entry in an *n*-deep FIFO is below a threshold. To broadcast the high traffic mode, the bus access module 410 sends a signal to the instruction request bus 225, which stores a traffic mode bit in the flip-flops 215a-e as a control bit sent to the processor clusters 210a-d. The bus access module 410 also sends synchronizing information such as a count or reset information. The processor clusters 210a-d, in response, send instruction requests during scheduled times, such as during an offset to the count.

[0047] If the average does not exceed 620 the limit, the bus access module 410 broadcasts 640 a low traffic mode to the processor clusters 210a-d. In this case, the processor clusters 210a-d can send instruction requests whenever they detect an empty slot. For example, the processor clusters 210a-d, can check a bit in associated flip-flops 215a-d that indicates whether an instruction request is already stored, whether the stored instruction request is busy, etc.

[0048] FIG. 7 is a flow chart illustrating the method 520 for resolving conflicts between instruction requests according to one embodiment of the present invention. One of ordinary skill in the art will recognize variations of resolving conflicts between conflicting instruction requests within the scope of the present invention. In FIG. 7, the memory access module 420 receives 710 instruction requests from the processor clusters 210a-d. If there are any conflicting requests 720, and the conflicting requests do not have the same priority 730, the memory access module 420 selects 740 the high priority instruction request. In an embodiment of two-priority levels, the low priority instruction is incremented, and possibly saturated, to a higher priority instruction request and recirculated around the instruction request bus 225.

[0049] If there are conflicting requests 720 that have the same priority 750, the memory access module 420 selects 750 an instruction request using round-robin arbitration. However, if the instruction requests do not conflict 720, the memory access module 420 forwards the maximum possible instruction requests to the instruction memory 230 without intervention.

IV. Summary

[0050] Advantageously, the present invention provides a processor 110 with an instruction memory hierarchy 300 to efficiently distribute packet processing instructions between a plurality of processing units 110 having multithreading capability. An instruction request arbiter 220 controls submissions to the instruction requests bus 225 by broadcasting a traffic mode and resolves conflicts between instruction requests by a priority associated with the instruction requests.